Alternating-Time Stream Logic for Multi-agent Systems

Sascha Klüppelholz and Christel Baier*

Technische Universität Dresden, Institut für Theoretische Informatik, Germany
{klueppel,baier}@tcs.inf.tu-dresden.de

Abstract. Constraint automata have been introduced to provide a compositional, operational semantics for the exogenous coordination language Reo, but they can also serve interface specification for components and an operational model for other coordination languages. Constraint automata have been used as basis for equivalence checking and model checking temporal logical properties. The main contribution of this paper is to reason about the local view and interaction and cooperation facilities of individual components or coalitions of components by means of a multi-player semantics for constraint automata. We introduce a temporal logic framework that combines classical features of alternating-time logic (*ATL*) for concurrent games with special operators to specify the observable data flow at the I/O-ports of components. Since constraint automata support any kind of synchronous and asynchronous peer-to-peer communication, the resulting game structure is non-standard and requires a series of nontrivial adaptations of the *ATL* model checking algorithm.

1 Introduction

In the last decade several models and specification languages for formal reasoning about the middle-ware layer of software have been developed. Such coordination models consist of ad-hoc libraries of functions providing higher-level inter-process communication support in parallel and especially distributed applications. They aim at a clean separation between individual software components and their interactions within their overall software organization. Our approach is inspired by the coordination language Reo [2], which provides the *glue-code* to coordinate components in an exogenous manner. In this paper we use constraint automata, which have been introduced as an operational semantics for Reo [6]. Constraint automata provide a specification formalism for both, the glue-code (e.g. given as a (Reo) network, or another (channel-based) coordination mechanism) and the behavioral interfaces of components, and can serve to formalize the overall behavior of the composite system. Constraint automata capture any kind of synchronous and asynchronous peer-to-peer communication including data-dependencies of I/O-operations. The syntax of constraint automata is similar to ordinary labeled transition systems and related models, such as timed port automata [15], I/O-automata [20], and interface automata [10]. The differences are mainly based on the fact that constraint automata support any kind of channel-based communication. An extensive discussion on the differences and similarities can be found in [6].

* The authors are supported by the DFG-NWO project SYANCO and the EU project CREDO.

D. Lea and G. Zavattaro (Eds.): COORDINATION 2008, LNCS 5052, pp. 184–198, 2008.

The purpose of this paper is to provide a multi-agent semantics for constraint automata and an alternating-time temporal logic to specify and verify the components considered as individual players of a multi-agent game. The connected components are the individual players and the network sets up the rules how those players interact with each other. The glue-code might be seen as a complex set of social laws [13,24] the players have to stick to. Constraint automata, interpreted as multi-player game structures, are a special type of concurrent games. The specific challenges of an alternating time approach are caused by the very special mixture of asynchrony and synchrony, mutual dependencies of I/O-operations and data-dependencies. In each state, several concurrent I/O-operations can be enabled, but only some of them might be available once a player refuses some synchronization or declares conditions on the data values accepted on his input ports or on his pending write operations. Furthermore, constraint automata can contain some internal nondeterminism, which yields a rather complex and nonstandard concurrent game structure. We are not aware of any other paper that treats alternating-time aspects for such concurrent games, where the enabledness and also the effect of a concurrent I/O-operation highly depends on the choices of the other players. Our approach allows us to check whether or not some coalition of agents has a strategy to achieve a common goal, no matter how the opponents behave, or which internal nondeterministic choices were made. In contrast to standard concurrent games, see e.g. [1,9], in our approach a coalition's strategy may select sets of I/O-operations or even refuse any I/O-operations.

For specifying and analyzing the local views and interaction possibilities of (coalitions of) agents, we introduce an alternating-time logic, called alternating-time stream logic (*ASL*). The logic *ASL* is a *CTL*-like branching-time logic which combines the features of standard *ATL* [1] with the operators of *BTSL* [18]. The logic *BTSL* has been introduced as a temporal logic for reasoning about (Reo) networks. Beside the standard modalities of *CTL* [8], *BTSL* supports the specification of the observable data flow at the I/O-ports of channels and components by means of regular expressions. The focus of *ATL* is to ask for the existence (and absence) of a coalition's strategy to achieve (avoid respectively) a specific temporal goal once the behavior for each of the components is specified.

For a simple example, we regard a ticket vending machine, which consists of a number of components (e.g. *I/O-device, clock, destination, price, payment, and printer*). The exact behavior of the components might be specified in terms of constraint automata. *ASL* can be used to formalize the property stating that the user (possibly together with some other component like the *clock*) can find a way to trick the other players and get a ticket without paying. A dual *ASL* property would state that no matter what strategy the opponents use, the coalition of opponents will not have a chance to avoid that sending the *cancel* signal always resets all components to their initial configuration.

As a first step we assume *perfect recall* on the systems history and *perfect information* on the global state of the system. This interpretation of constraint automata as a multi-player game is consistent with the standard semantics of *ATL* and adequate if the strategies are viewed as a central control that is aware of all activities in the system.

Our approach differs from other *ATL*-like approaches for concurrent multi-player games in various aspects. First, our nonstandard game structure (see explanations above)

requires a revised notion of strategies for (coalitions of) components. Second, since components may refuse any further interaction from some moment on, the concept of finite runs and fairness plays a crucial role in the logic *ASL*. To reason about liveness properties we need an adaption of the standard notion of strong (process) fairness. Our notion of fairness is not a requirement for strategies, but formalizes the ability of certain strategies of a component C to enforce infinite data flow at the I/O-ports of C. Third, *ASL* provides special operators to reason about the observable data flow at the I/O-ports of the components and the nodes of the given network. To the best of our knowledge, such operators have not yet been investigated in the context of alternating-time game models.

Organization. Section 2 gives a brief introduction to constraint automata. In section 3 we provide the multi-player semantics for constraint automata and introduce the notion of a strategy and its runs. Section 4 introduces the temporal logic *ASL* and presents corresponding model checking algorithms. Section 5 introduces fairness assumptions to *ASL* model checking, before section 6 concludes the paper. An extended technical report including the proofs and other technical material is available on the web [19].

2 Constraint Automata (CA)

This section summarizes the main concepts of CA. We slightly depart from the syntax of CA as introduced in [6] and deal with transitions $q \xrightarrow{c} p$, where c is a *concurrent I/O-operation*, i.e., c consists of a (possibly empty) node-set $N \subseteq \mathcal{N}$ together with data items for each $A \in N$ that are written or received at node A. In the moment where c is executed there is no data flow at the nodes $A \in \mathcal{N} \setminus N$.

Concurrent I/O-operations and I/O-streams. Let $\mathcal{N}$ be a finite, nonempty set of nodes. We define a concurrent I/O-operation as a function $c : \mathcal{N} \to \mathit{Data} \cup \{\bot\}$, where the symbol $\bot$ means "undefined". We write $\mathsf{Nodes}(c)$ for the set of nodes $A \in \mathcal{N}$ such that $c(A) \in \mathit{Data}$, where *Data* is the data domain. For technical reasons, we also allow the *empty* concurrent I/O-operation $c_\emptyset$ with $\mathsf{Nodes}(c_\emptyset) = \emptyset$. It represents any internal step of some component or a non-observable step, where data flow appears at some hidden (invisible) nodes only. We refer to CIO as the set of all concurrent I/O-operations (including $c_\emptyset$). As we suppose $\mathcal{N}$ and *Data* to be finite, the set CIO of concurrent I/O-operations is finite as well. When reasoning about the data flow in a Reo network we will also need a special symbol $\surd$ that indicates that data flow has stopped. $\mathsf{CIO}_\surd$ stands for $\mathsf{CIO} \cup \{\surd\}$.

Definition 1 (Constraint automata [6]). A constraint automaton (CA) is a tuple

$$\mathcal{A} = \langle Q, \mathcal{N}, \longrightarrow, Q_0, \mathit{AP}, L \rangle,$$

where Q is a finite and nonempty set of states, $\mathcal{N}$ a finite set of nodes, $\longrightarrow$ is a subset of $Q \times \mathsf{CIO} \times Q$ called the transition relation of $\mathcal{A}$, $Q_0 \subseteq Q$ a nonempty set of initial states, *AP* a finite set of atomic propositions, and $L : Q \to 2^{\mathit{AP}}$ a labeling function. We write $q \xrightarrow{c} p$ instead of $(q,c,p) \in \longrightarrow$. Furthermore, we define the set of all I/O-operations enabled in q as $\mathsf{CIO}(q) \stackrel{\text{def}}{=} \{ c \in \mathsf{CIO} : q \xrightarrow{c} p \text{ for some } p \in Q \}$.

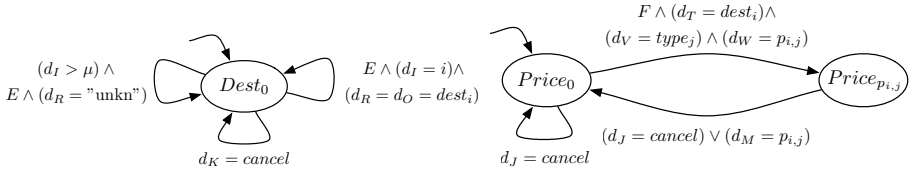
Intuitively, the nodes correspond to the I/O-ports of the components. For the pictures of CAs we shall use symbolic representations of the transition relation by combining transitions with the same starting and target state. For this purpose, we use I/O-constraints, i.e., propositional formulas in positive normal form that stand for sets of concurrent I/O-operations. The I/O-constraints may impose conditions on the nodes that may or may not be involved and on the data items written on or read from them.

I/O-constraints (IOC). The abstract syntax of I/O-constraints is given by the grammar:

$$ioc ::= tt \mid ff \mid A \mid \neg A \mid (d_{A_1},\ldots,d_{A_k}) \in D \mid ioc_1 \wedge ioc_2 \mid ioc_1 \vee ioc_2$$

where $A \in \mathcal{N}$, $A_1,\ldots,A_k$ are pairwise distinct nodes in $\mathcal{N}$ and $D \subseteq Data^k$. The meaning of an I/O-constraint *ioc* is a subset $CIO(ioc)$ of CIO defined in the obvious way. We often use simplified notations for the IOCs of the form $(d_{A_1},\ldots,d_{A_k}) \in D$. E.g., the notation $d_A = d_B$ is a shorthand for $(d_A, d_B) \in \{(d_1, d_2) \in Data^2 : d_1 = d_2\}$, while $A \wedge (d_B \in P)$ stands for the set $\{c \in \mathsf{CIO} : \{A, B\} \subseteq \mathsf{Nodes}(c) \wedge c(B) \in P\}$.

Example 1 (CA). The following two CAs realize possible implementations for the *destination* component with node set $\mathcal{N}_D = \{E, I, K, O, R\}$ and *price* component with node set $\mathcal{N}_P = \{F, J, M, T, V, W\}$ of the ticket vending machine. Both components are allowed to operate if and only if some data flow occurs on their synchronization ports E and F respectively. In the picture below we use a parameterized representation for states.



The *destination* component simultaneously reads some destination id (variable $\mathtt{i}$) on its input port I and writes the destination string (variable $\mathtt{dest_i}$) to the *I/O-device* using port R and its output port O. If the destination number given is too large, i.e., it exceeds a certain maximum μ, the *I/O-device* gets a message that the selected destination is unknown. The *price* component receives two integer values at its input ports T and V for the destination (variable $\mathtt{dest_i}$) and ticket type (variable $\mathtt{type_j}$) and sends the corresponding price (variable $\mathtt{p_{i,j}}$) first to the *I/O-device* using port W and in a second step to the *payment* component using port M. Both automata accept a *cancel* signal at any state and reset to their initial configuration.

Terminal States. A state q is called *terminal* if data flow may stop in state q. This is the case if all enabled concurrent I/O-operations require some activity of a component connected to a sink or source node. Formally, state q is said to be terminal if for all concurrent I/O-operations c that are enabled in state q, the node-set $\mathsf{Nodes}(c)$ is non-empty. Stated differently, state q is terminal iff $c_\emptyset \notin \mathsf{CIO}(q)$. Note that data flow does not need to stop in terminal states. Instead data flow continues if there is an enabled concurrent I/O-operation c where the involved components agree on interacting with each other by means of performing the write and read operation specified by c. For each

non-terminal node q, an invisible transition is enabled, i.e., we have $c_\emptyset \in \mathsf{CIO}(q)$. This I/O-operation does not require any interaction with the components that are connected to the sink and source nodes and will fire, unless another transition is taken.

Executions, Completeness, Paths, I/O-streams. An *execution* in $\mathcal{A}$ is a finite or infinite sequence built by instances of consecutive transitions: $\eta = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \dots$

$$\text{where } q_0, q_1, \dots \in Q,\ c_1, c_2, \dots \in \mathsf{CIO},\ \text{and } q_i \xrightarrow{c_{i+1}} q_{i+1} \text{ for all } i \geqslant 0.$$

To reason about "maximal" behaviors of CAs we introduce the notions of complete executions and paths. An execution is said to be *complete* if it is either infinite or it is finite and ends in a terminal state. A *path* of $\mathcal{A}$ is either an infinite execution or arises from a finite complete execution by adding a special transition symbol $\surd$ to denote termination. More precisely, the finite paths have the form $\pi = q_0 \xrightarrow{c_1} \dots \xrightarrow{c_n} q_n \xrightarrow{\surd} q_n$ where q_n is terminal. In the sequel, we shall use the symbol η for executions and the symbol π to range over paths. We write $\mathsf{Paths}(q)$ to denote the set of all paths starting in q and $\mathit{Exec}_{\mathrm{fin}}(q)$ for the set of all finite executions starting in q. The length $|\pi|$ of a path π is the total number of transitions taken in π (including the pseudo-transition with label $\surd$). Thus, the length of an infinite path is ∞, while the length of a finite path π as above is $n+1$. Let $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \dots$ be a path and $0 \leqslant n < |\pi|$. Then $\pi \downarrow n$ denotes the prefix of path π with length n, i.e., $\pi \downarrow n \stackrel{\text{def}}{=} q_0 \xrightarrow{c_1} \dots \xrightarrow{c_n} q_n$ is an execution, while for $n = |\pi|$ we have that $\pi \downarrow n = \pi$ is still a path. The *I/O-stream* $\mathit{ios}(\eta)$ of a finite execution η is the word over CIO that is obtained by taking the projection to the labels of the transitions. That is, if $\eta = q_0 \xrightarrow{c_1} \dots \xrightarrow{c_n} q_n$ then $\mathit{ios}(\eta) \stackrel{\text{def}}{=} c_1 \dots c_n$. Similarly, the associated I/O-stream for a finite path $\pi = q_0 \xrightarrow{c_1} \dots \xrightarrow{c_n} q_n \xrightarrow{\surd} q_n$ is defined by $\mathit{ios}(\pi) \stackrel{\text{def}}{=} c_1 \dots c_n \surd$. Let $\mathit{IOS} = \mathsf{CIO}^* \cup \mathsf{CIO}^* \surd$ denote the set of all I/O-streams.

3 Constraint Automata as Multi-player Games

In this section we introduce a game-theoretical interpretation for CA. The players are the individual components using (a)synchronous peer-to-peer communication. Each of the players has control over his I/O-behavior at its interface nodes. A player might refuse some or even any synchronization operation with other players. As in ordinary *ATL*, players might build arbitrary coalitions to achieve a certain common goal including a specific temporal behavior. A coalition of players induces a set of controllable nodes $N \subseteq \mathcal{N}$, the union of all controllable coalition nodes, for which the players might try to develop a common strategy to achieve their objective(s). Intuitively, an N-strategy takes the history of the system formalized by a finite execution as input, (i.e., we suppose here perfect recall) and declare the conditions under which the N-agents (members of the coalition) are willing to cooperate with each other and their opponents. For instance, an N-strategy might offer to write data value 0 at a source node $A \in N$, but refuse to write data value 1. The general notion of N-strategies also permits to couple such constraints for the offered I/O-operations at the N-nodes with conditions on the IOCs at the nodes in $\mathcal{N} \setminus N$. Furthermore, an N-strategy might suggest the N-agents to

refuse any participation in concurrent I/O-operations. The special symbol *stop* will be used for this purpose.

Definition 2 (Strategy). Let $\mathcal{A}$ be a CA as before and let N be a node-set such that $N \subseteq \mathcal{N}$. An N-strategy is a function

$$\mathfrak{S} : Exec_{\text{fin}}(\mathcal{A}) \to 2^{\mathsf{CIO} \cup \{stop\}},$$

assigning to any finite execution η a set $\mathfrak{S}(\eta)$ consisting of I/O-operations $c \in \mathsf{CIO}$ or the special symbol *stop* such that if $c \in \mathsf{CIO}$ and $\mathsf{Nodes}(c) \cap N = \emptyset$ then $c \in \mathfrak{S}(\eta)$.

The intuitive meaning of the condition required for an N-strategy asserts that the N-nodes are not in the position to refuse an I/O-operation c where none of the N-nodes is involved. In particular, invisible I/O-operations (i.e., concurrent I/O-operations with the empty node-set) cannot be ruled out by an N-strategy. A possible refinement for the notion of a strategy would be to allow components to restrict their write operations only and not to cut down any input provided at their boundary nodes. Given an N-strategy $\mathfrak{S}$, the $\mathfrak{S}$-paths are those paths in $\mathcal{A}$, where each of the I/O-operations performed is accepted at any time by the N-nodes and their strategy $\mathfrak{S}$.

Notation 3 ($\mathfrak{S}$-executions, $\mathfrak{S}$-completeness, $\mathfrak{S}$-paths). Let $\mathfrak{S}$ be an N-strategy and $\eta = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ a finite or infinite execution in $\mathcal{A}$. Then, η is called a $\mathfrak{S}$-*execution* if for any position $i \in \mathbb{N}$ with $i < |\eta|$ we have $c_{i+1} \in \mathfrak{S}(\eta \downarrow i)$. A finite $\mathfrak{S}$-execution η of length n is called $\mathfrak{S}$-*complete* if the last state q_n of η is terminal and at least one of the following two conditions holds:

(i) $stop \in \mathfrak{S}(\eta)$ or (ii) there is no $c \in \mathsf{CIO}(q_n) \cap \mathfrak{S}(\eta \downarrow n)$ such that $\mathsf{Nodes}(c) \subseteq N$

The first condition indicates that refusing any data flow on the N-nodes is a potential behavior under strategy $\mathfrak{S}$, while the second indicates the possibility for the opponents to do the same on their part (i.e. refusing any synchronization on the $\mathcal{N} \setminus N$ nodes). Furthermore, each infinite $\mathfrak{S}$-execution is said to be $\mathfrak{S}$-complete. A $\mathfrak{S}$-*path* denotes any infinite $\mathfrak{S}$-execution or any finite path $\pi = q_0 \xrightarrow{c_1} \ldots \xrightarrow{c_n} q_n \xrightarrow{\checkmark} q_n$, where $\pi \downarrow n$ is a $\mathfrak{S}$-complete $\mathfrak{S}$-execution. We write $\mathsf{Paths}(q, \mathfrak{S})$ to denote all $\mathfrak{S}$-paths starting in q. Similarly, $Exec_{\text{fin}}(q, \mathfrak{S})$ denotes the set of all finite $\mathfrak{S}$-executions from q.

Notation 4 (Memoryless, finite-memory strategies). An N-strategy $\mathfrak{S}$ is called *memoryless* if $\mathfrak{S}(\eta) = \mathfrak{S}(\eta')$ for all finite executions η and η' that end in the same state. Memoryless strategies can be seen as functions $\mathfrak{S} : Q \to 2^{\mathsf{CIO} \cup \{stop\}}$. Obviously, memoryless strategies are special instances of *finite-memory* strategies, i.e., strategies that make their decisions on the basis of a finite automaton rather than the full history.

4 Alternating-Time Stream Logic (ASL)

To reason about the components from a game-theoretic point of view, we introduce *alternating-time stream logic* (*ASL*) which is inspired by alternating-time temporal logic

(*ATL*) [1]. *ASL* extends *BTSL* [18] to state the possibility for components to cooperate in such way that a certain temporal property or property on the observable data flow holds. *ASL* is a branching time logic with state and path formulas. The state formula fragment is as in *ATL*, but adapted to the CA framework where the alternating-time quantifiers range over the strategies of certain node-sets. Intuitively, these node-sets stand for the interface nodes of one or more components. The existential quantifier $\mathbb{E}_N$ is used to indicate that the components with sink and source nodes in N have a strategy ensuring a certain condition, no matter how the other components connected to the nodes in $\mathcal{N} \setminus N$ behave. The universal quantifier $\mathbb{A}_N$ is dual and serves to state that the components providing the write and read actions at the N-nodes cannot avoid that a certain condition holds. The syntax of the *ASL* path formulas is the same as in *BTSL* and uses the standard until- and release operator, but replaces the standard next modality $\bigcirc$ with special operators $\langle\!\langle \alpha \rangle\!\rangle$ and $[\![\alpha]\!]$ to impose conditions on the I/O-streams of finite executions. In path formulas of the type $\langle\!\langle \alpha \rangle\!\rangle \Phi$ or $[\![\alpha]\!]\Phi$, the formula Φ is a state formula while α is a regular expression that stands for a regular language over the alphabet $\mathsf{CIO}_{\surd}$. This type of formulas is inspired by propositional dynamic logic [12], extended temporal logic [23], and timed scheduled data stream logic [3].

4.1 Syntax and Standard Semantics of ASL

In the sequel, we assume a fixed, non-empty and finite node-set $\mathcal{N}$. Furthermore, let *AP* be non-empty and finite set of atomic propositions, which can be viewed as conditions on the states of the automaton. In case of the CA modeling a FIFO-channel an atomic proposition might state that all buffer cells are empty or that the first buffer cell contains a value d in some set $P \subseteq \mathit{Data}$.

Regular I/O-stream Expressions. The abstract syntax of regular I/O-stream expressions, briefly called stream expressions, is given by the following grammar:

$$\alpha \ ::= \ \mathit{ioc} \ \Big| \ \surd \ \Big| \ \alpha^* \ \Big| \ \alpha_1;\alpha_2 \ \Big| \ \alpha_1 \cup \alpha_2$$

where *ioc* ranges over all IOCs. Any stream expression represents a regular set of I/O-streams. The formal definition of the regular languages $\mathit{IOS}(\alpha) \subseteq \mathit{IOS}$ is defined by structural induction. $\mathit{IOS}(\mathit{ioc})$ is the set consisting of the I/O-streams of length 1 given by *ioc*, i.e., $\mathit{IOS}(\mathit{ioc}) \stackrel{\text{def}}{=} \mathit{CIO}(\mathit{ioc})$. Similarly, $\mathit{IOS}(\surd)$ is the singleton set consisting of the I/O-stream $\surd$. Union ($\cup$) has its standard meaning: $\mathit{IOS}(\alpha_1 \cup \alpha_2) \stackrel{\text{def}}{=} \mathit{IOS}(\alpha_1) \cup \mathit{IOS}(\alpha_2)$, while Kleene star ($*$) and concatenation (;) have to ensure that the special termination symbol $\surd$ can only appear at the end of an I/O-stream:

$$\mathit{IOS}(\alpha^*) \stackrel{\text{def}}{=} \{\varepsilon\} \cup \bigcup_{n \geqslant 1} \{\sigma_1 \ldots \sigma_n : \sigma_i \in \mathit{IOS}(\alpha) \cap \mathsf{CIO}^*, i = 1, \ldots, n-1, \sigma_n \in \mathit{IOS}(\alpha)\}$$

$$\mathit{IOS}(\alpha_1;\alpha_2) \stackrel{\text{def}}{=} \{\sigma_1 \surd : \sigma_1 \surd \in \mathit{IOS}(\alpha_1)\} \cup \{\sigma_1\sigma_2 : \sigma_1 \in \mathit{IOS}(\alpha_1) \cap \mathsf{CIO}^*, \sigma_2 \in \mathit{IOS}(\alpha_2)\}$$

Syntax of ASL. State-formulas (denoted by capital greek letters Φ, Ψ) and path-formulas (denoted by small greek letters φ, ψ) of *ASL* are built by the following grammar:

$$\Phi ::= \mathsf{true} \mid \mathfrak{a} \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \mathbb{E}_{\mathsf{N}}\varphi$$
$$\varphi ::= \langle\!\langle\alpha\rangle\!\rangle\Phi \mid [\![\alpha]\!]\Phi \mid \Phi_1 \,\mathtt{U}\, \Phi_2 \mid \Phi_1 \,\mathtt{R}\, \Phi_2$$

where $\mathsf{N} \subseteq \mathcal{N}$, $\mathfrak{a} \in AP$ and α is a regular I/O-stream expression. The quantifier $\exists$ in the syntax of *ASL* state formulas is the standard existential path quantifier of *CTL* and ranges over all paths, while the operator $\mathbb{E}_{\mathsf{N}}$ corresponds an existential quantification over all N-strategies. The dual operator $\mathbb{A}_{\mathsf{N}}\varphi$ stating that no strategy for the nodes in N can avoid φ to hold is defined by:

$$\mathbb{A}_{\mathsf{N}}\langle\!\langle\alpha\rangle\!\rangle\Phi \stackrel{\text{def}}{=} \neg\mathbb{E}_{\mathsf{N}}[\![\alpha]\!]\neg\Phi \qquad \mathbb{A}_{\mathsf{N}}(\Phi_1 \,\mathtt{U}\, \Phi_2) \stackrel{\text{def}}{=} \neg\mathbb{E}_{\mathsf{N}}(\neg\Phi_1 \,\mathtt{R}\, \neg\Phi_2)$$
$$\mathbb{A}_{\mathsf{N}}[\![\alpha]\!]\Phi \stackrel{\text{def}}{=} \neg\mathbb{E}_{\mathsf{N}}\langle\!\langle\alpha\rangle\!\rangle\neg\Phi \qquad \mathbb{A}_{\mathsf{N}}(\Phi_1 \,\mathtt{R}\, \Phi_2) \stackrel{\text{def}}{=} \neg\mathbb{E}_{\mathsf{N}}(\neg\Phi_1 \,\mathtt{U}\, \neg\Phi_2)$$

In an analogous way, the universal *CTL*-path quantifier $\forall$ can be derived by duality from $\exists$. (Alternatively, $\forall\varphi$ can be defined by $\mathbb{E}_{\emptyset}\varphi$.) Other boolean connectives, like disjunction or implication, are obtained in the obvious way. In the following we shortly write $\mathbb{E}_{\mathsf{A}}\varphi$ for $\mathbb{E}_{\{\mathsf{A}\}}\varphi$ and $\mathbb{A}_{\mathsf{A}}\varphi$ for $\mathbb{A}_{\{\mathsf{A}\}}\varphi$.

ASL path formulas are interpreted over paths in a CA. The modalities $\mathtt{U}$ and $\mathtt{R}$ denote the ordinary until-operator and release-operator, respectively. The eventually and always operator are obtained in the usual way by $\Diamond\Phi \stackrel{\text{def}}{=} (\mathsf{true}\,\mathtt{U}\,\Phi)$ and $\Box\Phi \stackrel{\text{def}}{=} (\mathsf{false}\,\mathtt{R}\,\Phi)$.

The intended meaning of $\langle\!\langle\alpha\rangle\!\rangle\Phi$ is that it holds for a path π iff π has a finite prefix generating an α-stream and Φ holds for the state reached afterwords. $[\![\alpha]\!]\Phi$ is the dual operator of $\langle\!\langle\alpha\rangle\!\rangle\Phi$ and holds for a path π iff for all finite prefixes of π generating an α-stream, formula Φ holds for the last state of the prefix. The standard *next* operator is derived from the path formula $\bigcirc\Phi \stackrel{\text{def}}{=} \langle\!\langle tt\rangle\!\rangle\Phi$, which asserts the occurrence for some (non-observable) data flow. Recall that $IOS(tt) = CIO(tt) = \mathsf{CIO}$. Thus, $\bigcirc\Phi$ holds for all paths where the underlying execution has at least one transition and Φ holds afterwords. The presence of some observable data flow can be expressed by $\langle\!\langle \mathsf{A}_1 \vee \ldots \vee \mathsf{A}_n\rangle\!\rangle\mathsf{true}$, where $\mathcal{N} = \{\mathsf{A}_1, \ldots, \mathsf{A}_n\}$. The path formula $[\![tt^*; \surd]\!]\mathsf{false}$ is characteristic for the infinite paths, while $\langle\!\langle tt^*; \surd\rangle\!\rangle\mathsf{true}$ holds exactly for the finite paths. The terminal states are characterized by the state formula $\exists\langle\!\langle\surd\rangle\!\rangle\mathsf{true}$, while $\forall\langle\!\langle\surd\rangle\!\rangle\mathsf{true}$ is satisfied in exactly those states where no concurrent I/O-operation is enabled. *ASL* state formulas are the same as in *BTSL* except for the $\mathbb{E}_{\mathsf{N}}$-operator (and its dual).

For an intuitive example, consider a FIFO-channel with source node A and sink node B. Then the *ASL* state formulas $\mathbb{E}_{\mathsf{A}}\Box\mathsf{empty}$, $\mathbb{E}_{\mathsf{A}}\Box(\mathsf{buffer} \neq 0)$, $\mathbb{A}_{\mathsf{B}}\Diamond\mathsf{empty}$ and $\mathbb{A}_{\mathsf{B}}\Box\mathsf{empty}$ do hold, where $(\mathsf{buffer} \neq 0)$ states that either the buffer is empty or contains a data value different from zero. In case of the ticket vending machine we may ask whether the user (possibly in coalition with other components) controlling three boundary nodes $\mathsf{N} = \{\mathsf{C}, \mathsf{D}, \mathsf{P}\}$ (for the *cancel* signal, data items, and payment) has a strategy to get a ticket without paying, i.e. if state formula $\mathbb{E}_{\{\mathsf{C},\mathsf{D},\mathsf{P}\}}\langle\!\langle\neg pay^*\rangle\!\rangle ticket\_printed$ holds. A dual *ASL* property states that all components except the user respect the *cancel* signal and reset to their initial configuration. This can be expressed by $\mathbb{A}_{\mathcal{N}\setminus\mathsf{N}}[\![tt^*; \mathsf{C}]\!]initconf$.

Standard Semantics of ASL. Let $\mathcal{A}$ be a CA and π a path in $\mathcal{A}$. The satisfaction relation $\models$ for *ASL* state formulas is defined by structural induction as shown below:

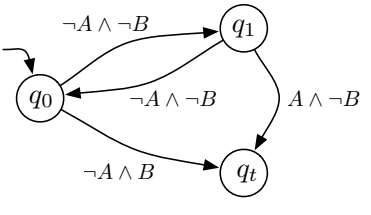
$$
\begin{array}{lll}
q \models \mathsf{true} & & \\
q \models a & \text{iff} & a \in L(q) \\
q \models \Phi_1 \wedge \Phi_2 & \text{iff} & q \models \Phi_1 \text{ and } q \models \Phi_2 \\
q \models \neg\Phi & \text{iff} & q \not\models \Phi \\
q \models \exists\varphi & \text{iff} & \text{there exists } \pi \in \mathsf{Paths}(q) \text{ such that } \pi \models \varphi \\
q \models \mathbb{E}_N\varphi & \text{iff} & \text{there is an N-strategy } \mathfrak{S} \text{ such that:} \\
 & & \quad \text{for all } \pi \in \mathsf{Paths}(q,\mathfrak{S}) : \pi \models \varphi
\end{array}
$$

The satisfaction relation $\models$ for *ASL* path-formuls and the path π in $\mathcal{A}$ as follows:

$$
\begin{array}{lll}
\pi \models \langle\!\langle \alpha \rangle\!\rangle \Phi & \text{iff} & \text{there exists } n \in \mathbb{N} \text{ such that } 0 \leqslant n \leqslant |\pi| \text{ and} \\
 & & \quad ios(\pi \downarrow n) \in IOS(\alpha) \text{ and } q_n \models \Phi \\
\pi \models [\![\alpha]\!] \Phi & \text{iff} & \text{for all } n \in \mathbb{N} \text{ such that } 0 \leqslant n \leqslant |\pi| \text{ we have:} \\
 & & \quad ios(\pi \downarrow n) \in IOS(\alpha) \text{ implies } q_n \models \Phi \\
\pi \models \Phi_1 \,\mathrm{U}\, \Phi_2 & \text{iff} & \text{there exists } n \in \mathbb{N} \text{ such that } 0 \leqslant n < |\pi| \text{ where} \\
 & & \quad q_n \models \Phi_2 \text{ and } q_i \models \Phi_1 \text{ for } 0 \leqslant i < n \\
\pi \models \Phi_1 \,\mathrm{R}\, \Phi_2 & \text{iff} & \text{at least one of the following conditions (i) or (ii) holds:} \\
 & & \quad \text{(i) for all } n \in \mathbb{N} \text{ with } 0 \leqslant n < |\pi| \text{ we have: } q_n \models \Phi_2 \\
 & & \quad \text{(ii) there exists some } n \in \mathbb{N} \text{ with } 0 \leqslant n \leqslant |\pi| \text{ such that:} \\
 & & \quad\quad q_n \models \Phi_1 \text{ and } q_i \models \Phi_2 \text{ for } 0 \leqslant i \leqslant n
\end{array}
$$

Given a state q and a *ASL* path formula φ, an N-strategy $\mathfrak{S}$ is called *winning* for the tuple $\langle q, \varphi \rangle$ if φ holds for all $\mathfrak{S}$-paths starting in q. Thus, $q \models \mathbb{E}_N\varphi$ iff there exists a winning N-strategy for $\langle q, \varphi \rangle$. For the derived operator $\mathbb{A}_N$ we get that $q \models \mathbb{A}_N\varphi$ iff for all N-strategies $\mathfrak{S}$ there exists $\pi \in \mathsf{Paths}(q, \mathfrak{S})$ such that $\pi \models \varphi$, i.e. there is no winning strategy for $\langle q, \varphi \rangle$.

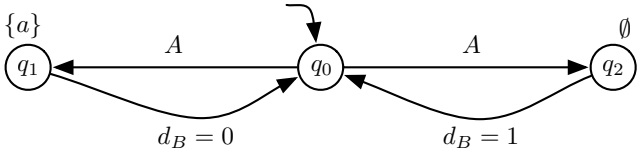
Example 2 (ASL state formulas). The CA with node set $\mathcal{N} = \{A, B\}$ depicted below fulfills the following state formula $\mathbb{A}_A \Diamond \neg \exists \bigcirc \mathsf{true}$, stating that an agent controlling A only cannot avoid that a terminal state q_t will eventually be reached.



The multi-player game associated with a CA and an *ASL* path formula is *not determined*. In fact, there are path formulas φ such that neither the N-agents have a winning strategy for φ nor does the opponents (i.e., the $\mathcal{N} \setminus$ N-agents) have a strategy to ensure that φ does not hold. The reason for this is that the internal nondeterminism can yield the possibility to generate paths where φ holds and paths where φ does not hold.

In particular, the *ASL* state formulas $\mathbb{E}_{\mathsf{N}}\varphi$ and $\mathbb{A}_{\mathcal{N}\setminus\mathsf{N}}\varphi$ are *not* equivalent[1] and $q \models \mathbb{E}_{\mathsf{N}}\varphi$ implies $q \models \mathbb{A}_{\mathcal{N}\setminus\mathsf{N}}\varphi$ holds for all states $q \in Q$, but not vice versa. A simple example illustrating this fact is the following CA with node-set $\mathcal{N} = \{A, B\}$.

Example 3 (Internal nondeterminism).



Assume that $a \in AP$ is an atomic proposition which holds in q_1 only, i.e. $L(q_1) = \{a\}$ and $L(q_2) = \emptyset$. Since the internal nondeterminism decides whether q_1 or q_2 will be selected as successor state of q_0 when A fires, neither A can enforce nor B can avoid that q_1 will be entered in the next step. Thus, we have $q_0 \models \mathbb{A}_{\mathsf{B}}\bigcirc a$ and $q_0 \not\models \mathbb{E}_{\mathsf{A}}\bigcirc a$.

4.2 ASL Model Checking

The model checking problem for *ASL* asks whether, for a given CA $\mathcal{A}$ and *ASL* state formula Φ, all initial states q_0 of $\mathcal{A}$ satisfy Φ. The main procedure for *ASL* model checking follows the standard approach for *CTL*-like branching-time logics [8] and recursively calculates the satisfaction sets $Sat(\Psi) = \{q \in Q : q \models \Psi\}$ for all sub-formulas Ψ of Φ. The treatment of the *BTSL*-fragment of *ASL* is the same as for *BTSL* (see [18]). The only interesting part is how to calculate $Sat(\mathbb{E}_{\mathsf{N}}\varphi)$ for an *ASL* path formulas φ and node-set $\mathsf{N} \subseteq \mathcal{N}$. The essential ingredient for this is the predecessor operator $Pre(\mathsf{P}, \mathsf{N})$ which is defined as the set of all states $q \in Q$ such that the N-nodes have a strategy which guarantees to move within one step to a state in P.

Definition 5 (Predecessors). Let $\mathsf{P} \subseteq Q$ and $\mathsf{N} \subseteq \mathcal{N}$ a node-set. Then, $Pre(\mathsf{P}, \mathsf{N})$ denotes the set of all states $q \in Q$ such that the following two conditions hold:

(i) for all $c \in \mathsf{CIO}(q)$ such that $\mathsf{Nodes}(c) \cap \mathsf{N} = \emptyset$ we have $Post[c](q) \subseteq \mathsf{P}$
(ii) there exists a $c \in \mathsf{CIO}(q)$ such that $\mathsf{Nodes}(c) \subseteq \mathsf{N}$ and $Post[c](q) \subseteq \mathsf{P}$

where $Post[c](q) \stackrel{\text{def}}{=} \{p \in Q : q \xrightarrow{c} p\}$.

Condition (i) is needed to ensure that no uncontrollable transition (from the view of the N-agents) leads to a state outside of P, while condition (ii) asserts the existence of at least one concurrent I/O-operation that can be enforced by the N-agents and certainly leads to a state in P. In fact we have $Pre(\mathsf{P}, \mathsf{N}) = \left\{ q \in Q : q \models \mathbb{E}_{\mathsf{N}}\bigcirc\mathsf{P} \right\}$.

As for standard *CTL* (and *ATL*), the semantics of the until and release operator have a fixed point characterization. The set $Sat(\mathbb{E}_{\mathsf{N}}(\Phi_1 \mathrel{\mathsf{U}} \Phi_2))$ is the least fixpoint, while the set $Sat(\mathbb{E}_{\mathsf{N}}(\Phi_1 \mathrel{\mathsf{R}} \Phi_2))$ is the greatest fixpoint of the following operators $2^Q \to 2^Q$:

$$\mathsf{P} \mapsto Sat(\Phi_2) \cup (Pre(\mathsf{P}, \mathsf{N}) \cap Sat(\Phi_1)) \quad \text{(until)}$$
$$\mathsf{P} \mapsto Sat(\Phi_2) \cap (Pre(\mathsf{P}, \mathsf{N}) \cup Sat(\Phi_1)) \quad \text{(release)}$$

[1] The same observation holds for *ATL** interpreted over concurrent games, but for other reasons.

Hence, in *ASL* with the standard semantics we have the following *expansion laws*:

$$\mathbb{E}_{\mathsf{N}}(\Phi_1 \,\mathtt{U}\, \Phi_2) \equiv \Phi_2 \vee (\Phi_1 \wedge \mathbb{E}_{\mathsf{N}} \bigcirc \mathbb{E}_{\mathsf{N}}(\Phi_1 \,\mathtt{U}\, \Phi_2)) \tag{1}$$

$$\mathbb{E}_{\mathsf{N}}(\Phi_1 \,\mathtt{R}\, \Phi_2) \equiv \Phi_2 \wedge (\Phi_1 \vee \mathbb{E}_{\mathsf{N}} \bigcirc \mathbb{E}_{\mathsf{N}}(\Phi_1 \,\mathtt{R}\, \Phi_2)) \tag{2}$$

where $\equiv$ denotes equivalence of *ASL* state formulas. On the basis of (1) and (2), we obtain that for winning objectives formalized by *ASL* path formulas φ of the form $(\Phi_1 \,\mathtt{U}\, \Phi_2)$ or $(\Phi_1 \,\mathtt{R}\, \Phi_2)$, memoryless strategies are sufficient and the satisfaction set $\mathit{Sat}(\mathbb{E}_{\mathsf{N}}\, \varphi)$ can be computed by means of the standard procedures to compute least and greatest fixed points of monotonic operators. The algorithms for until and release including the proof of correctness can be found in the technical report [19]. For *ASL* state formulas of the form $\mathbb{E}_{\mathsf{N}} \langle\!\langle \alpha \rangle\!\rangle \Phi$ or $\mathbb{E}_{\mathsf{N}} [\![\alpha]\!] \Phi$, we follow an automata-theoretic approach which resembles the standard automata-based *LTL* model checking procedure and relies on a representation of α by means of a finite automaton $\mathcal{Z}$ and a graph analysis of the product $\mathcal{A} \bowtie \mathcal{Z}$. As α is roughly an ordinary regular expression, we can apply standard methods to generate a deterministic finite automata $\mathcal{Z}$ over the alphabet $\mathsf{CIO}_{\surd}$ such that the accepted language of $\mathcal{Z}$ agrees with $\mathit{IOS}(\alpha)$.

Let $\mathcal{Z} = (\mathsf{Z}, \mathsf{CIO}_{\surd}, \delta, \mathsf{Z}_0, \mathsf{Z}_{\mathsf{F}})$, i.e., Z stands for the state space, z_0 the initial state, Z_{F} for the set of final (accept) states and $\delta : \mathsf{Z} \times \mathsf{CIO}_{\surd} \to \mathsf{Z}$ for the transition function. In fact, beside the special $\surd$-transitions, $\mathcal{Z}$ can be viewed as a CA where the set Z_{F} plays the role of the labeling function which separates the final states from the non-final states. Due to the special role of the symbol $\surd$ (which can only appear at the end of a word in $\mathit{IOS}(\alpha)$), we can assume that there are special states $z_{\mathit{accept}} \in \mathsf{Z}_{\mathsf{F}}$ and $z_{\mathit{reject}} \in \mathsf{Z} \setminus \mathsf{Z}_{\mathsf{F}}$ such that each $\surd$-transition leads to one of the states z_{accept} or z_{reject} and that the states z_{accept} or z_{reject} cannot be entered via any other symbol. Given $\mathcal{A}$ and $\mathcal{Z}$, we built the product $\mathcal{A} \bowtie \mathcal{Z}$, similar to the product of finite automata and the join operator for CAs [6], but with a special treatment of the pseudo-transitions with label $\surd$. In fact, the product construction we use here differs from those used in the *BTSL* model checking procedure [18] since in the context of the $\mathbb{E}_{\mathsf{N}}$-operator we have to incorporate the possibilities of the N-agents to enforce termination. Formally, we define the CA $\mathcal{A} \bowtie_{\mathsf{N},\Phi} \mathcal{Z}$ as follows:

$$\mathcal{A} \bowtie_{\mathsf{N},\Phi} \mathcal{Z} \stackrel{\text{def}}{=} (\mathsf{S}, \mathcal{N} \cup \{\mathsf{A}_{\mathit{stop}}\}, \longrightarrow, \mathsf{S}_0, \mathit{AP}', \mathsf{L}').$$

The state space S is $\mathsf{Q} \times \mathsf{Z}$ and $\mathsf{A}_{\mathit{stop}}$ is a new node-name (not contained in $\mathcal{N}$). This new node is supposed to be controllable. (Thus, for $\mathcal{A} \bowtie_{\mathsf{N},\Phi} \mathcal{Z}$ we will ask for $(\mathsf{N} \cup \{\mathsf{A}_{\mathit{stop}}\})$-strategies rather than N-strategies.) The initial states are given by

$$\mathsf{S}_0 = \big\{ \langle \mathsf{q}, \mathsf{z}_0 \rangle : \ \mathsf{q} \in \mathsf{Q}_0 \big\}.$$

The atomic propositions and labeling function in $\mathcal{A} \bowtie_{\mathsf{N},\Phi} \mathcal{Z}$ are given by the set $\mathit{AP}' = \{\mathsf{a}_\Phi, \mathit{accept}\}$, where $\mathsf{a}_\Phi \in \mathsf{L}'(\langle \mathsf{q}, \mathsf{z} \rangle)$ iff $\mathsf{q} \models \Phi$ and $\mathit{accept} \in \mathsf{L}'(\langle \mathsf{q}, \mathsf{z} \rangle)$ iff $\mathsf{z} \in \mathsf{Z}_{\mathsf{F}}$. The transitions in $\mathcal{A} \bowtie_{\mathsf{N},\Phi} \mathcal{Z}$ are obtained by the following synchronization rule for concurrent I/O-operations $\mathsf{c} \in \mathsf{CIO}$ (i.e., $\mathsf{c} \neq \surd$), state q in $\mathcal{A}$, and state $\mathsf{z} \in \mathsf{Z} \setminus \{z_{\mathit{accept}}, z_{\mathit{reject}}\}$:

$$\frac{\mathsf{q} \xrightarrow{\mathsf{c}}_{\mathcal{A}} \mathsf{q}' \ \wedge \ \mathsf{z} \xrightarrow{\mathsf{c}}_{\mathcal{Z}} \mathsf{z}'}{\langle \mathsf{q}, \mathsf{z} \rangle \xrightarrow{\mathsf{c}} \langle \mathsf{q}', \mathsf{z}' \rangle} \tag{3}$$

where we use the subscript $\mathcal{A}$ for the transition relations in $\mathcal{A}$. In addition, we have the following rules for each terminal state q in $\mathcal{A}$ and state $z \in Z \setminus \{z_{accept}, z_{reject}\}$ where c_{stop} is a concurrent I/O-operation with $\mathsf{Nodes}(c_{stop}) = \{A_{stop}\}$ and $c_{stop}(A_{stop})$ is an arbitrary element from the data domain *Data*:

$$\frac{\neg\exists c \in \mathsf{CIO}(q) \text{ s.t. } \mathsf{Nodes}(c) \subseteq N \wedge c_\emptyset \notin \mathsf{CIO}(q)}{\langle q, z\rangle \xrightarrow{c_\emptyset} \langle q, \delta(z, \surd)\rangle} \tag{4}$$

$$\frac{\exists c \in \mathsf{CIO}(q) \text{ s.t. } \mathsf{Nodes}(c) \cap N \neq \emptyset \wedge c_\emptyset \notin \mathsf{CIO}(q)}{\langle q, z\rangle \xrightarrow{c_{stop}} \langle q, \delta(z, \surd)\rangle} \tag{5}$$

Rule (4) formalizes the fact that if q is terminal (i.e., $c_\emptyset \notin \mathsf{CIO}(q)$) and there is no $c \in \mathsf{CIO}(q)$ such that $\mathsf{Nodes}(c) \subseteq N$ then the opponents of the N-agents may refuse any write or read operation and can therefore enforce data flow to stop. This is modeled in the product by a transition with the label $c_\emptyset$. Rule (5) stands for the fact that whenever q is a terminal node for which some concurrent I/O-operation c is enabled where the N-nodes are involved then the N-agents might decide not to participate in any further I/O-operation. This is modeled in the product by a transition with the label c_{stop} where the new node A_{stop} is supposed to be controllable. We obtain the following two lemmas for *ASL* state formulas of the form $\mathbb{E}_N\langle\!\langle\alpha\rangle\!\rangle\Phi$ and $\mathbb{E}_N[\![\alpha]\!]\Phi$.

Lemma 1. Let $\mathcal{A}$ be a CA, $\mathcal{Z} = (Z, \mathsf{CIO}_\surd, \delta, Z_0, Z_F)$ a DFA for α, q in $\mathcal{A}$, node-sets $N \subseteq \mathcal{N}$ and *ASL* state formulas Φ. Then, the following statements are equivalent:

(a) $q \models \mathbb{E}_N\langle\!\langle\alpha\rangle\!\rangle\Phi$
(b) $\langle q, z_0\rangle \models \mathbb{E}_{N\cup\{A_{stop}\}}\Diamond(a_\Phi \wedge \mathit{accept})$
(c) There exists a finite-memory N-strategy $\mathfrak{S}$ for $\mathcal{A}$ that is winning for $\langle q, \langle\!\langle\alpha\rangle\!\rangle\Phi\rangle$

Lemma 2. Let $\mathcal{A}$ be a CA, $\mathcal{Z} = (Z, \mathsf{CIO}_\surd, \delta, Z_0, Z_F)$ a DFA for α, q in $\mathcal{A}$, node-sets $N \subseteq \mathcal{N}$ and *ASL* state formulas Φ. Then, the following statements are equivalent:

(a) $q \models \mathbb{E}_N[\![\alpha]\!]\Phi$
(b) $\langle q, z_0\rangle \models \mathbb{E}_{N\cup\{A_{stop}\}}\Box(\mathit{accept} \rightarrow a_\Phi)$
(c) there exists a finite memory N-strategy $\mathfrak{S}$ which is winning for $\langle q, [\![\alpha]\!]\Phi\rangle$

Thanks to lemmas 1 and 2 the satisfaction sets $\mathit{Sat}(\mathbb{E}_N\langle\!\langle\alpha\rangle\!\rangle\Phi)$ and $\mathit{Sat}(\mathbb{E}_N[\![\alpha]\!]\Phi)$ can be computed by means of a reduction to the model checking problem for the $\mathbb{E}_N$-operator in combination with the eventually- and always-modalities. More precisely, we first have to construct a DFA $\mathcal{Z}$ for α, then built the product $\mathcal{A} \bowtie_{N,\Phi} \mathcal{Z}$ and finally apply the algorithm for until and release respectively, to compute the satisfaction sets for $\mathbb{E}_{N\cup\{A_{stop}\}}\Diamond(a_\Phi \wedge \mathit{accept})$ and $\mathbb{E}_{N\cup\{A_{stop}\}}\Box(\mathit{accept} \rightarrow a_\Phi)$ in the product. Furthermore the memoryless $(N \cup \{A_{stop}\})$-strategies for the product yield finite-memory winning N-strategies in $\mathcal{A}$ for the objectives $\langle\!\langle\alpha\rangle\!\rangle\Phi$ and $[\![\alpha]\!]\Phi$, respectively.

Assuming that $\mathit{Sat}(\Phi)$ has already been computed the time complexity for computing $\mathit{Sat}(\mathbb{E}_N\langle\!\langle\alpha\rangle\!\rangle\Phi)$ and $\mathit{Sat}(\mathbb{E}_N[\![\alpha]\!]\Phi)$ is linear in the size of CA $\mathcal{A}$ and the DFA $\mathcal{Z}$ for α (which can be exponential in the length of α). However, when restricting to the *ATL*-fragment of *ASL* which just uses the standard path modalities $\mathtt{U}$, $\mathtt{R}$ and $\bigcirc$, but

not $\langle\!\langle\alpha\rangle\!\rangle$ or $[\![\alpha]\!]$, then the worst complexity of the *ASL* model checking algorithm is the same as for standard *ATL*, i.e., linear in the size of $\mathcal{A}$ and the length of the formula.

We conclude this section by a simple observation concerning the case that α is a $\surd$-free expression (i.e., does not contain a subexpression of the form $\beta;\surd$). In fact, for $\surd$-free expressions, the "best" strategy for the N-agents to ensure $[\![\alpha]\!]\Phi$ is to stop the data flow whenever possible. This is formalized in the following lemma.

Lemma 3 (Winning strategies for $\surd$-free expressions). Let $\mathfrak{S}_{stop}$ be the memoryless N-strategy given by $\mathfrak{S}_{stop}(q) = \{stop\} \cup \{c \in \mathsf{CIO} : \mathsf{Nodes}(c) \cap N = \emptyset\}$ for all states q. Then, for each $\surd$-free stream expression α and state q we have:

$$q \models \mathbb{E}_N [\![\alpha]\!]\Phi \text{ iff } \mathfrak{S}_{stop} \text{ is winning for } \langle q, [\![\alpha]\!]\Phi\rangle.$$

Thus, if α is $\surd$-free then the set $Sat(\mathbb{E}_N [\![\alpha]\!]\Phi)$ can be computed by considering the sub-automaton $\mathcal{A}'$ of $\mathcal{A}$ that results by the memoryless strategy $\mathfrak{S}_{stop}$ and then computing the satisfaction set for $Sat_{\mathcal{A}'}(\forall [\![\alpha]\!]\Phi)$ in $\mathcal{A}'$. This can be done by means of a *BTSL* model checker [18].

5 ASL with Fairness

The concept of fairness serves to rule out pathological behaviors, where certain liveness properties are violated, although they are supposed to hold [14]. The nondeterminism within our multi-player setting demand for some *ASL* fairness assumptions. To illustrate the need for some fairness assumptions, we reuse the deadlock example (2). One would expect that the *ASL* state formula $\mathbb{E}_B \Diamond \neg \exists \bigcirc \mathsf{true}$ would be fulfilled, since the memoryless strategy $\mathfrak{S}$, which tries to write on B whenever q_0 is reached during an execution should be winning for $\langle q_0, \Diamond \neg \exists \bigcirc \mathsf{true}\rangle$. But

$$\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} q_0 \xrightarrow{c_1} \ldots \in \mathsf{Paths}(q_0, \mathfrak{S}) \text{ and } \pi \not\models \neg \exists \bigcirc \mathsf{true}.$$

The goal of this section is to introduce some fairness assumptions to exclude such undesirable behaviors from our observations.

Definition 6 ($\langle N, \mathfrak{S}\rangle$-fairness). Let $\mathcal{A} = \langle Q, \mathcal{N}, \longrightarrow, Q_0, AP, L\rangle$ be a CA, $N \subseteq \mathcal{N}$ a node-set, $\mathfrak{S}$ an N-strategy, and $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} \ldots$ a $\mathfrak{S}$-path in $\mathcal{A}$. Then π is called *(strongly)* $\langle N, \mathfrak{S}\rangle$*-fair* if either π is finite or for all $c \in \mathsf{CIO}$ we have:

$$\overset{\infty}{\exists}\, i \geqslant 0.\ c \in \mathsf{CIO}(q_i) \cap \mathfrak{S}(\pi \downarrow i) \text{ and } \mathsf{Nodes}(c) \subseteq N \text{ implies } \overset{\infty}{\exists}\, i \geqslant 0.\ c_i = c,$$

where $\overset{\infty}{\exists}\, i$ means "there exists infinitely many i". We write $\mathsf{FairPaths}_{\langle N,\mathfrak{S}\rangle}(q)$ for all $\langle N, \mathfrak{S}\rangle$-fair paths starting in q and $\mathsf{FairPaths}_{\langle N,\mathfrak{S}\rangle}(\mathcal{A})$ for the set of $\langle N, \mathfrak{S}\rangle$-fair paths.

In the above example, $\pi = q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} q_0 \xrightarrow{c_1} \ldots \notin \mathsf{FairPaths}_{\langle\{B\},\mathfrak{S}\rangle}(q_0)$ because $\mathfrak{S}$ is willing to write infinitely often on B, but no write operation is ever executed. The semantics of the fair *ASL* path formulas is the same as for *ASL* without fairness (see section 4.1).

The semantics for fair *ASL* state formulas also corresponds to the one without fairness except for:

$$q \models_{\mathsf{fair}} \mathbb{E}_N \varphi \text{ iff there is an } N\text{-strategy } \mathfrak{S} \text{ s.t. for all } \pi \in \mathsf{FairPaths}_{\langle N,\mathfrak{S}\rangle}(q) : \pi \models \varphi$$

The underlying model checking algorithms need to be modified and now rely on the bottom up computation of the sets $Sat_{\mathsf{fair}}(\Psi) = \{q \in Q \mid q \models_{\mathsf{fair}} \Psi\}$ for all subformulas Ψ. The computation for $Sat_{\mathsf{fair}}(\mathbb{E}_N(\Phi_1 \,\mathrm{R}\, \Phi_2))$ does not involve any modification at all, as shown in the following lemma.

Lemma 4 (Release with fairness). Let $\mathcal{A}$ be a CA, $N \subseteq \mathcal{N}$ a node-set, $q \in Q$ a state in $\mathcal{A}$ and Φ_1, Φ_2 *ASL* state formulas. Then $q \models_{\mathsf{fair}} \mathbb{E}_N(\Phi_1 \,\mathrm{R}\, \Phi_2)$ iff $q \models \mathbb{E}_N(\Phi_1 \,\mathrm{R}\, \Phi_2)$.

The computation of $Sat_{\mathsf{fair}}(\mathbb{E}_N(\Phi_1 \,\mathrm{U}\, \Phi_2))$ relies on an iterative SCC-calculation in subgraphs of $\mathcal{A}$. The following lemma emerges that the remaining fair computation of $Sat_{\mathsf{fair}}(\mathbb{E}_N \langle\!\langle \alpha \rangle\!\rangle \Phi)$ and $Sat_{\mathsf{fair}}(\mathbb{E}_N [\![\alpha]\!] \Phi)$ can be reduced to eventually and always in the product $\mathcal{A} \bowtie \mathcal{Z}$.

Lemma 5 (Fairness for ASL I/O-stream expression formulas). Let $\mathcal{A}$ be a CA, $N \subseteq \mathcal{N}$ a node-set, α a regular I/O-stream expression, $\mathcal{Z}$ a deterministic CA for α, and let Φ be *ASL* state formula. Then, the following observation holds for all states $q \in Q$.

i) $q \models_{\mathsf{fair}} \mathbb{E}_N \langle\!\langle \alpha \rangle\!\rangle \Phi$ in $\mathcal{A}$ iff $\langle q, z_0 \rangle \models_{\mathsf{fair}} \mathbb{E}_{N \cup \{A_{stop}\}} \Diamond (accept \wedge \mathfrak{a}_\Phi)$ in $\mathcal{A} \bowtie \mathcal{Z}$.
ii) $q \models_{\mathsf{fair}} \mathbb{E}_N [\![\alpha]\!] \Phi$ iff $\langle q, z_0 \rangle \models_{\mathsf{fair}} \mathbb{E}_{N \cup \{A_{stop}\}} \Box (accept \rightarrow \mathfrak{a}_\Phi)$ in $\mathcal{A} \bowtie \mathcal{Z}$.

6 Conclusion and Future Work

This paper introduces a framework to verify alternating-time properties for a multi-player games derived from CA. The introduced concurrent game semantics captures any complex behavior caused by synchronous and asynchronous peer-to-peer communication, mutual dependencies of I/O-operations and also data-dependencies. Since this game structure is non-standard it takes numerous nontrivial adaptations of the *ATL* model checking algorithm. In future work we will drop our assumption on *perfect information* and *perfect recall* to switch to a more realistic view for exogenous coordination taking the local view [5,7,16,17,21,11,22] into account. In future work we will consider *observation-based strategies* in case of *incomplete information*.

Apart from asking for the existence or absence of a winning strategy for a temporal property the question might raise, if there is a way of connecting the components to make this property hold. This directly leads to the controller synthesis problem where if possible a controlling CA is put in parallel with the other components to ensure the intended behavior. One step further we would like to build the Reo network which glues those components the intended way by using the synthesis approach described in [4].

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM 49, 672–713 (2002)
2. Arbab, F.: Reo: A channel-based coordination model for component composition. Mathematical Structures in Computer Science 14(3), 329–366 (2004)

3. Arbab, F., Baier, C., de Boer, F., Rutten, J.J.M.M.: Models and temporal logics for timed component connectors. In: Proc. of SEFM, pp. 198–207. IEEE CS Press, Los Alamitos (2004)
4. Arbab, F., Baier, C., de Boer, F., Rutten, J.J.M.M., Sirjani, M.: Synthesis of Reo circuits for implementation of component connector automata specifications. In: Jacquet, J.-M., Picco, G.P. (eds.) COORDINATION 2005. LNCS, vol. 3454, Springer, Heidelberg (2005)
5. Azhar, S., Peterson, G.L., Reif, J.H.: On multiplayer non-cooperative games of incomplete information: Part 1&2. Technical report, Durham, NC, USA (1991)
6. Baier, C., Sirjani, M., Arbab, F., Rutten, J.J.M.M.: Modeling component connectors in Reo by constraint automata. In: Science of Computer Programming 61, pp. 75–113 (2006)
7. Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.F.: Algorithms for omega-regular games with imperfect information. CoRR, abs/0706.2619 (2007)
8. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM TOPLAS 8(2), 244–263 (1986)
9. de Alfaro, L., Henzinger, T.A.: Concurrent omega-regular games. In: Proc. of LICS, pp. 141–154 (January 2000)
10. de Alfaro, L., Henzinger, T.A.: Interface automata. In: FSE Proc., pp. 109–120. ACM Press, New York (2001)
11. de Wulf, M., Doyen, L., Raskin, J.-F.: A lattice theory for solving games of imperfect information. In: Hespanha, J.P., Tiwari, A. (eds.) HSCC 2006. LNCS, vol. 3927, pp. 153–168. Springer, Heidelberg (2006)
12. Fischer, M.J., Ladner, R.J.: Propositional dynamic logic of regular programs. Journal of Computer and System Science 8, 194–211 (1979)
13. Fitoussi, D., Tennenholtz, M.: Choosing social laws for multi-agent systems: minimality and simplicity. Artif. Intell. 119(1-2), 61–101 (2000)
14. Francez, N.: Fairness. Springer, Heidelberg (1986)
15. Grosu, R., Rumpe, B.: Concurrent timed port automata. Technical Report TUM-I9533, Techn. Univ. München (1995), `http://www4.informatik.tu-muenchen.de/reports/`
16. Hoek, W.v.d., Roberts, M., Wooldridge, M.: Knowledge and social laws. In: AAMAS, pp. 674–681 (2005)
17. Hoek, W.v.d., Wooldridge, M.: Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. Studia Logica 75(1), 125–157 (2003)
18. Klüppelholz, S., Baier, C.: Symbolic model checking for channel-based component connectors. In: Proc. of FOCLASA 2006. ENTCS, vol. 175(2), pp. 19–37 (2007)
19. Klüppelholz, S., Baier, C.: Alternating-Time Stream Logic for Multi-Agent Systems. Technical report, Technical University Dresden (2008), `http://wwwtcs.inf.tu-dresden.de/~klueppel/ASLKB2008.pdf`
20. Lynch, N., Tuttle, M.R.: An introduction to input/output automata. CWI Quarterly 2(3), 219–246 (1989)
21. Reif, J.H.: The complexity of two-player games of incomplete information. J. Comput. Syst. Sci. 29(2), 274–301 (1984)
22. Schobbens, P.Y.: Alternating-time logic with imperfect recall. In: Proc. of LCMAS. ENTCS, vol. 85(2), pp. 1–12 (2004)
23. Wolper, P.: Specification and synthesis of communicating processes using an extended temporal logic. In: Proc. of POPL, pp. 20–33 (1982)
24. Wooldridge, M.: Social laws in alternating time. In: Lomuscio, A., Nute, D. (eds.) DEON 2004. LNCS (LNAI), vol. 3065, p. 2. Springer, Heidelberg (2004)